

Assigning and examining variables

Scalars: (1×1 matrices)

```
>> a = 1 + 2 * (9 - 3)
```

```
a =
```

```
13
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x1	8	double array

```
Grand total is 1 elements using 8 bytes
```

```
>> a
```

```
a =
```

```
13
```

```
>>
```

3

EECS C145B/ BioE C165: Lecture 4

Introduction to MATLAB

Why MATLAB?

- Provides environment rich with tools for numerical analysis.
- Contains all basic functions for matrix, vector and scalar arithmetic, as well as advanced functions too numerous to mention.
- Provides extensive tools for visualizing data.
- Code written is completely transportable to different computing platforms upon which MATLAB runs.
- Usually more than 10 times faster to implement algorithms in MATLAB as compared to starting afresh using language such as C.
- MATLAB is interactive. Even in the middle of running a program, program execution can be stopped and the variables investigated and plotted.

1

Elementary vector operations

Vectors: ($N \times 1$ or $1 \times N$ matrices)

```
>> vec1 = [ 1 1.5 3 2 -1 1e-1]
```

```
vec1 =  
1.0000 1.5000 3.0000 2.0000 -1.0000 0.1000
```

```
>> size(vec1)
```

```
ans =  
1 6
```

```
>> vec2 = [ 1 3 5 7 9 11].'
```

```
vec2 =  
1  
3  
5  
7  
9  
11
```

```
>> vec1 + vec2
```

```
??? Error using ==> +  
Matrix dimensions must agree.
```

```
>> vec1 + vec2.'
```

```
ans =  
2.0000 4.5000 8.0000 9.0000 8.0000 11.1000
```

Disadvantages

- MATLAB code which is not **vectorized** is very slow. Need to avoid long **for** or **while** loops whenever possible.
- MATLAB is not appropriate for dealing with very large problems e.g. matrices that are hundreds of megabytes in size.
- MATLAB is proprietary and expensive. A user must buy MATLAB before they can run any programs written in MATLAB.

4

2

Elementary MATLAB functions

```
>> help sin
SIN      Sine.
        SIN(X) is the sine of the elements of X.

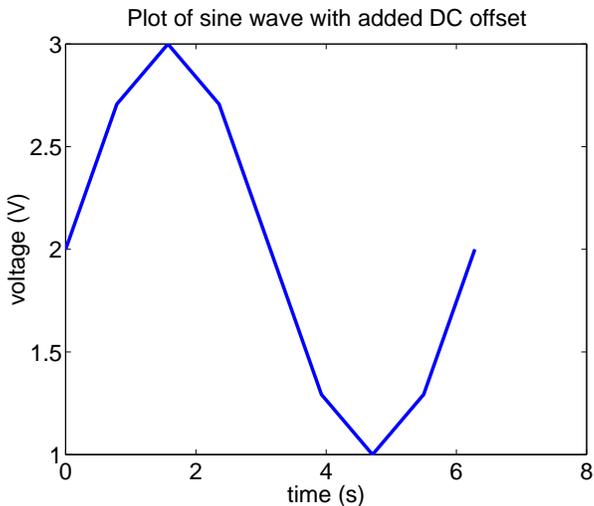
>> t = 0:(pi/4):(2*pi)
t =
Columns 1 through 7
    0    0.7854    1.5708    2.3562    3.1416    3.9270    4.7124
Columns 8 through 9
    5.4978    6.2832

>> f = sin(t) + 2
f =
Columns 1 through 7
    2.0000    2.7071    3.0000    2.7071    2.0000    1.2929    1.0000
Columns 8 through 9
    1.2929    2.0000

>> figure(1)
>> plot(t,f)
>> title('Plot of sine wave with added DC offset')
>> xlabel('time (s)')
>> ylabel('voltage (V)')
```

7

Resulting plot



8

Elementary matrix manipulation

```
>> myMatrix = [1 2 3;
               -1 -2 -3;
               .2 -.9 34]
myMatrix =
    1.0000    2.0000    3.0000
   -1.0000   -2.0000   -3.0000
    0.2000   -0.9000   34.0000

>> a = myMatrix(1,3)
a =
     3

>> myMatrix(2,:) = [999 888 777]
myMatrix =
    1.0000    2.0000    3.0000
   999.0000  888.0000  777.0000
    0.2000   -0.9000   34.0000

>> myMatrix .* myMatrix
ans =
    1.0e+05 *

    0.0000    0.0000    0.0001
    9.9800    7.8854    6.0373
    0.0000    0.0000    0.0116
```

5

```
>> ans(1,:)
ans =
     1     4     9

>> ans(2)
ans =
     4

>> inv(myMatrix)
ans =
   -0.7731    0.0018    0.0278
    0.8461   -0.0008   -0.0556
    0.0269    0.0000    0.0278

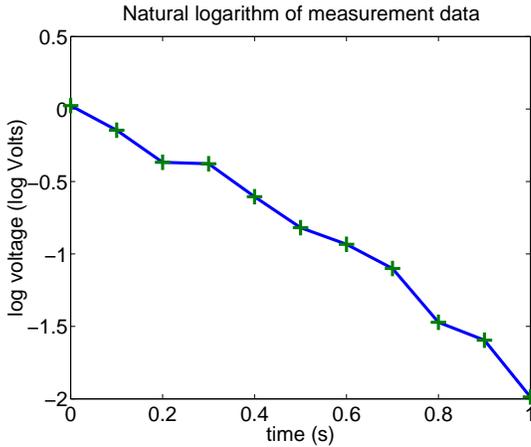
>> ans * myMatrix
ans =
    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

6

Curve-fitting example continued:

```
>> gLogged = log(g);
>> plot(t,gLogged, t, gLogged,'+')

>> % we will fit a straight line to the log of the measurements
>> % because linear fits are easy to do
```

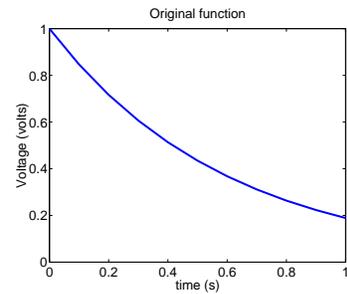


11

Curve-fitting example

```
>> t = 0:.1:1
t =
    Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
    Columns 8 through 11
         0.7000    0.8000    0.9000    1.0000

>> origFun = exp(-t/.6);
>> plot(t, origFun); % plots function in figure window
>> title('Original function')
>> print -djpeg origFun.jpg % make a jpeg version of you plot
                             % maybe to display on a web page
>> print -dps2 origFun.ps % print to PostScript Level 2 file.
                             % To see other format options,
                             % type: 'help print'
>> ! lpr -Plw105 origFun.ps % print to printer called 'lw105'
                             % in 105 Cory Hall.
                             % '!' means it's a host operating
                             % system shell command to execute
>> ! lp -dlw105 origFun.ps % Alternative print command for some
                             % unix systems e.g. HP
```



9

Casting the line fitting problem in terms of matrix operations:

Objective is to fit model: $y = c \exp^{-t/\tau}$ to the data

$$\text{simultaneous equations} \begin{cases} \log(y(t_0)) = mt_0 + \log(c) \\ \log(y(t_1)) = mt_1 + \log(c) \\ \log(y(t_2)) = mt_2 + \log(c) \\ \vdots \\ \log(y(t_{10})) = mt_{10} + \log(c) \end{cases}$$

$$\begin{bmatrix} \log(y(t_0)) \\ \log(y(t_1)) \\ \log(y(t_2)) \\ \vdots \\ \log(y(t_{10})) \end{bmatrix} = \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \\ t_2 & 1 \\ \vdots & 1 \\ t_{10} & 1 \end{bmatrix} \begin{bmatrix} m \\ \log(c) \end{bmatrix}$$

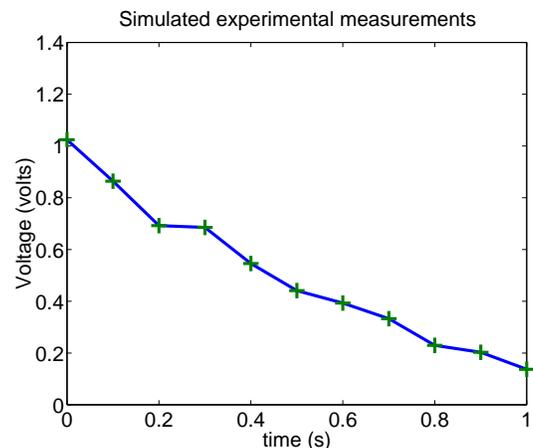
$$\begin{bmatrix} m \\ \log(c) \end{bmatrix} = \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \\ t_2 & 1 \\ \vdots & 1 \\ t_{10} & 1 \end{bmatrix}^{\dagger} \begin{bmatrix} \log(y(t_0)) \\ \log(y(t_1)) \\ \log(y(t_2)) \\ \vdots \\ \log(y(t_{10})) \end{bmatrix}$$

12

Curve-fitting example continued

```
>> stdDev = .05;
>> noise = randn(1,11) * stdDev % gaussian random number generator
noise =
    Columns 1 through 7
         0.0363   -0.0294   0.1092   -0.0068   0.0057   0.0533   0.0030
    Columns 8 through 11
        -0.0048   -0.0416   0.0147   -0.0668

>> g = origFun + noise;
>> plot(t,g, t,g,'+')
>> title('Simulated experimental measurements')
```



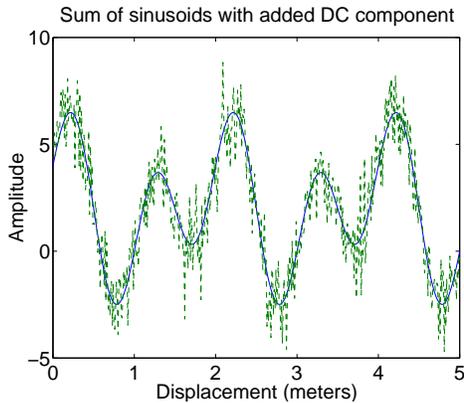
10

Convolution in 1D Smoothing noisy signals

```
>> t = 0:.01:5;
>> f1 = 1; % cycles/meter
>> f2 = .5; % cycles/meter
>> f = 3*sin(2*pi*f1*t) + 2*cos(2*pi*f2*t) +2;
>> rows = size(f,1);
rows =
     1

>> cols = size(f,2)
cols =
    501

>> stdDev = 1; % meter
>> fNoisy = f + ( randn(1,501) * stdDev );
>> plot(t,f, t,fNoisy,'--');
```



15

Curve-fitting example: MATLAB implementation

```
>> H = [t.' ones(11,1)]
H =
     0     1.0000
    0.1000     1.0000
    0.2000     1.0000
    0.3000     1.0000
    0.4000     1.0000
    0.5000     1.0000
    0.6000     1.0000
    0.7000     1.0000
    0.8000     1.0000
    0.9000     1.0000
    1.0000     1.0000

>> mLogc = pinv(H) * gLogged.' % Simply regard 'pinv'
% as a black box which
% calculates the 'inverse' of a
% rectangular matrix. It is
% a generalized inverse or
% pseudoinverse. You'll learn
% more about it later.

mLogc =

    -1.9318
     0.1058

>> timeConst = -1/mLogc(1)
timeConst =
     0.5177

>> intercept = exp(mLogc(2))
intercept =
     1.1116
```

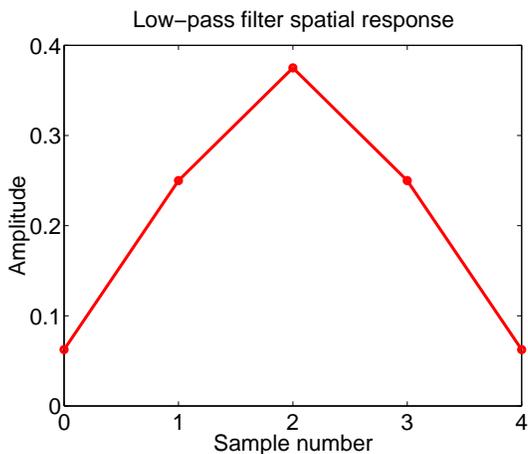
13

Convolution in 1D continued

```
>> filterKernel = [ 1 4 6 4 1 ] / 16; % approximation
% to gaussian
>> figure(2) % bring up a new,
% empty, figure window

>> sampleAxis = 0:1:4
sampleAxis =
     0     1     2     3     4

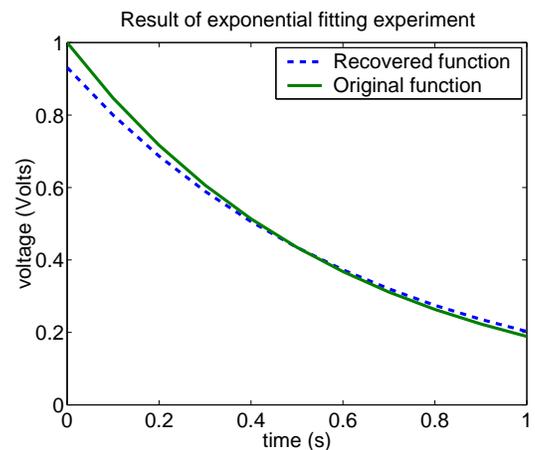
>> plot(sampleAxis, filterKernel, sampleAxis, filterKernel, 'o')
```



16

Plot result against original function

```
>> recoveredFun = intercept * exp(-t/timeConst);
>> plot(t,recoveredFun,'--', t, origFun);
>> legend('Recovered function', 'Original function')
```



14

Obtaining the DFT of signals

```
>> F0 = fft(f);
>> FN = fft(fNoisy);
>> FR = fft(filterOutput);
>> size(filterOutput)
ans =
     1    501

>> size(FR)
ans =
     1    501

>> freqAxisSpacing = 2*pi/501;
>> frequencyAxis = 0:freqAxisSpacing:2*pi-freqAxisSpacing;

>> subplot(3,1,1)
>> plot(frequencyAxis, log(abs(F0)))
>> title('Log magnitude of DFT of original signal')

>> subplot(3,1,2)
>> plot(frequencyAxis, log(abs(FN)))
>> title('Log magnitude of DFT of noisy signal')

>> subplot(3,1,3)
>> freqAxisSpacing2 = 2*pi/505;
>> frequencyAxis2 = 0:freqAxisSpacing2:2*pi-freqAxisSpacing2;
>> plot(frequencyAxis2, log(abs(FR)))
>> title('Log magnitude of DFT of filtered signal')
```

19

Convolution in 1D continued:

```
>> length(fNoisy)
ans =
    501

>> length(filterKernel)
ans =
     5

>> filterOutput = conv(fNoisy, filterKernel);

>> length(filterOutput)
ans =
    505

>> lengthOutput = length(fNoisy)+length(filterKernel)-1
lengthOutput =
    505

>> xaxisForOutput = 0:lengthOutput-1;

>> xaxisForInput = 0:length(f)-1;

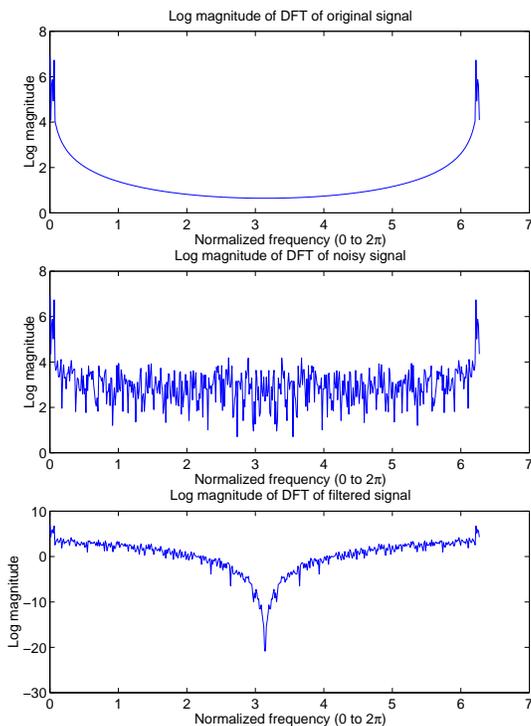
>> subplot(2,1,1) % 2 rows (1st arg) of subplots in figure
                 % 1 column (2nd arg) of plots
                 % want next plot to appear in first (3rd arg)
                 % of the two subplots

>> plot(xaxisForInput, fNoisy)

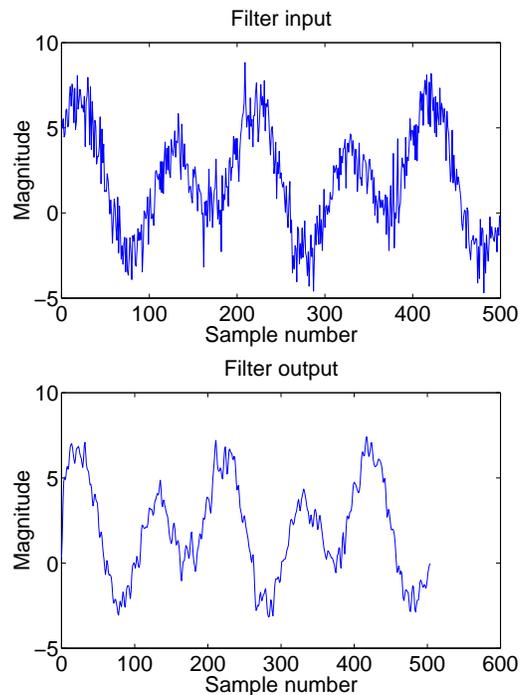
>> subplot(2,1,2)
>> plot(xaxisForOutput, filterOutput)
```

17

DFT log-magnitude plots



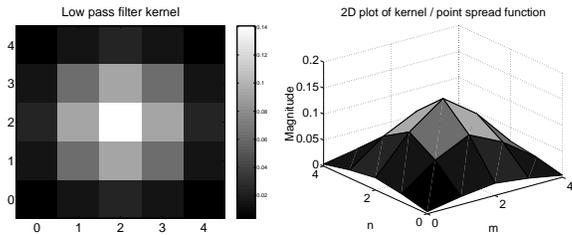
20



18

Convolution in 2D continued

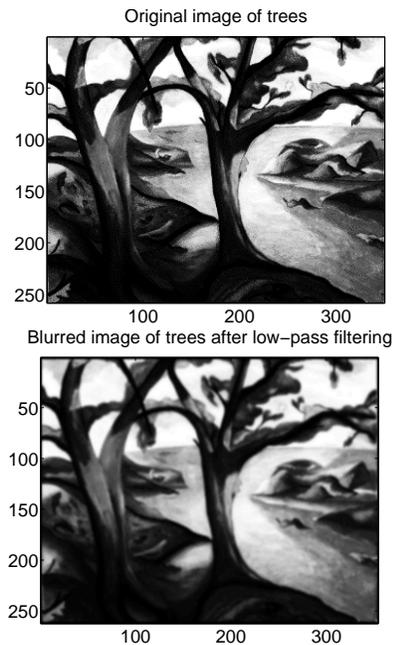
```
>> lowPassKernel = [1 4 6 4 1;
                    4 16 24 16 4;
                    6 24 36 24 6;
                    4 16 24 16 4;
                    1 4 6 4 1] / 256;
% Aside: Where does this kernel come from? 1D gaussian-
% approximating low pass filters of increasing length
% are obtained by repeatedly convolving [1 1]
% with itself eg. [1 2 1].' [1 3 3 1].'
% The outer products eg. [1 2 1].' * [1 2 1] give the
% corresponding family of 2D filters.
% The divisor 256 is for normalization (so that all entires
% add to 1), so that the filter does not modify the total
% energy of the image to which it is applied.
```



23

Convolution in 2D continued

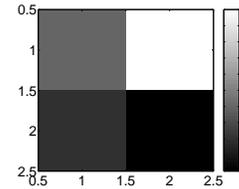
```
>> lowPassIm = conv2(X, lowPassKernel);
>> imagesc(lowPassIm) % displays lower image on this page
```



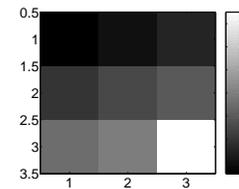
24

Introduction to images

```
>> A = [.2 .5;
        .1 0]
A =
    0.2000    0.5000
    0.1000         0
>> % two commands can be placed on 1 line separated by semicolon:
>> imagesc(A); colorbar
>> colormap(gray)
```



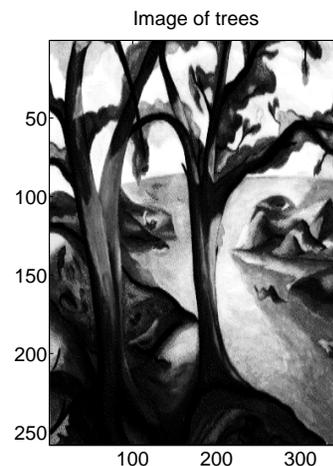
```
>> B = [ 1 2 3;
        4 5 6;
        7 8 15];
>> imagesc(B); colorbar
```



21

Convolution in 2D

```
>> load trees % loads binary file trees.mat containing
              % Matlab variables into workspace
>> whos
Name          Size          Bytes  Class
X             258x350       722400 double array
caption       1x66          132    char array
map          128x3         3072   double array
Grand total is 90750 elements using 725604 bytes
>> imagesc(X)
```



22